Paralelizace

SSPŠ
PVA4
2025 / 2026
Ing. Matěj Cajthaml

Úvodní terminologie

- Procesory
- jednojádrové
- vícejádrové
- hyper-threading, multi-threading a další

- Program
- Instrukce
- Proces
- Vlákno

- Program (instrukce) => proces => vlákno

Vlákna

- Thread
- Vlákna jednoho procesu vidí sdílené prostředky
- Mohou si do nich "šahat" a měnit je

- zásobník: lokální proměnné, návratová adresa
- přepínání: čítače instrukcí, registry, ...
- plánování: priorita, stav, ...
- id

Zpracování

- OS, resp. planovač
- Určuje jaké vlákno má jaký procesor a kdy

- Dle priority, jak dlouho neběželo, …
- Pokud se dokončí, nebo se něco stane, nebo běží dlouho
- Tak se přeruší

- Pokud nekončí, uloží se stav (registry, místo instrukce, …)

Tvorba vláken

- Systémové volání
- Většinou předáváme ukazatel na místo v programu (instrukci/funkci)

- fork, tvorba nového procesu
- Nesdílí data, tedy komunikuji mimo sebe, např. sockety

- Problém se sdílením

API v C#

- Třída System.Threading.Thread
- Určujeme funkci
- Různé metody

Ukázka / 1

```
1  using System.Threading;
2
3  Thread t = new Thread(run);
4
5  Console.WriteLine("Hello from the main thread");
6  t.Start();
7
8  public static void run() {
9      Console.WriteLine("Hello from the thread");
10 }
```

Ukázka / 2

```
1  using System.Threading;
2
3  Thread t1 = new Thread(() => run(32));
4  Thread t2 = new Thread(() => run(123));
5
6  Console.WriteLine("Hello from the main thread");
7
8  t1.Start();
9  t2.Start();
10
11 public static void run(int i) {
12     Console.WriteLine("Hello from the thread {0}", i);
13 }
```

Připojení se k vláknu

- Počkáme, až se vlákno dokončí
- .Join()
- Do té doby aktuální vlákno čeká

- Čekání vlákně => .Sleep()
- Pasivní čekání
- Aktivní čekání

Ukázka / 3

```
1  using System.Threading;
2
3  Thread t1 = new Thread(() => fact(5));
4  Thread t2 = new Thread(() => wait(5));
5
6  Console.WriteLine("Hello from the main thread");
7
8  t1.Start();
9  t2.Start();
10
11 t1.Join();
12
13 Console.WriteLine("Here we are again");
14
15 t2.Join();
```

Ukázka / 4

```
1  using System.Threading;
2
3  List<uint> results = new List<uint>();
4
5  Thread t1 = new Thread(() => start(100, results));
6  Thread t2 = new Thread(() => start(100, results));
7
8  t1.Start();
9  t2.Start();
10
11 t1.Join();
12 t2.Join();
13
14 Console.WriteLine("Results: {0}", string.Join(", ", results));
15 Console.WriteLine("Sorted: {0}", string.Join(", ", results.OrderBy(x => x)));
```

Chyby bez synchronizace

- Operace nejsou atomické
- Vlákna se přepínají, běží paralelně
- Vlákna si mohou přepsat data nebo pracovat s neplatnými daty

- Kritická sekce
- Více vláken může být v jedné sekci současně
- Vlákna se musí synchronizovat - kdo co udělal, kam co zapsat
- Spousta způsobů

Zámky

- Nejčastější operace
- Atomická - přístup do sekce má jen jedno vlákno
- Při přístupu se zamkne
- Při odchodu se odemkne
- Pokud je při příchodu již zamknuté, čeká se až se odemkne

- ReleaseMutex
- WaitOne

- Tisíce způsobů jak vyřešit spojení hodnot

Zámky / 1

```
1  using System.Threading;
2
3  List<uint> results = new List<uint>();
4
5  Mutex mutex = new Mutex();
6
7  Thread t1 = new Thread(() => start(100, results, mutex));
8  Thread t2 = new Thread(() => start(100, results, mutex));
9  t1.Start();
10 t2.Start();
11
12 t1.Join();
13 t2.Join();
14
15 Console.WriteLine("Results: {0}", string.Join(", ", results));
```

Zámky / 2

```
1  using System.Threading;
2
3  List<uint> results = new List<uint>();
4
5  Mutex mutex = new Mutex();
6
7  Thread t1 = new Thread(() => start(100, results, mutex));
8  Thread t2 = new Thread(() => start(100, results, mutex));
9  t1.Start();
10 t2.Start();
11
12 t1.Join();
13 t2.Join();
14
15 Console.WriteLine("Results: {0}", string.Join(", ", results));
```

OTÁZKA

Zjistěte si informace o semaforech, monitorech
a bariérách pro synchronizaci.

Chyby v paralelizaci

- Časově závislé chyby:
- vlákna se zapisují do sdílených dat, vlákna se přepínají, chyba závislá na čase
- Uváznutí (deadlock):
- vlákna čekají na událost, a událost může zavolat jen čekající vlákno
- Živé uváznutí (livelock):
- vlákna zpracovávají program, vlákna ale nemohou dokončit program
- Hladovění
- Řešení: správné pořadí, pečlivě navržené zamykání

Nejčastější využití

- Programy: main thread a UI thread
- Síťové servery: web, hry, ..., více vláken obsahuje část hračů
- Náročné výpočty a Multimédia
- I/O operace

Asynchronní programování

- Většinově jen jedno vlákno
- Uvnitř vlákna se umíme přepínat
- V praxi nějaké věci čekají - DB, I/O, ...
- A my nechceme aktivně čekat

- Můžeme během čekání zpracovávat jinou věc

- Často na bázi event loopu
- Jazyky často tuto smyčku nemají a musí se dodávat
- Často:
- async - funkce "může" trvat
- await - čekáme na výsledek async funkce

- Často čekání na čekání
- Promise/Tasks

Asynchronní / 1

```
interface Post {
  userId: number;
  id: number;
  title: string;
  body: string;
}

async function getPostData(): Promise<void> {
  const url = "https://jsonplaceholder.typicode.com/posts/1";

  try {
    const response = await fetch(url);

    if (!response.ok) {
      throw new Error(`Error: ${response.status}`);
```

Asynchronní / 2

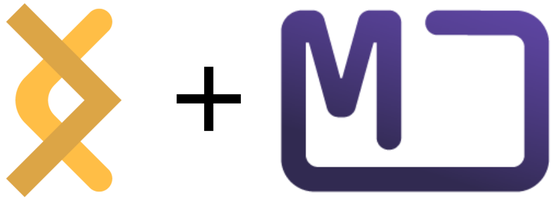
```rust
use reqwest;

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let url = "https://jsonplaceholder.typicode.com/posts/1";

    let response = reqwest::get(url).await?;

    if response.status().is_success() {
        let body = response.text().await?;
        println!("Data:\n{}", body);
    } else {
        println!("Error: {}", response.status());
    }

```

OTÁZKA

Může být asynchronní kód vícevláknový?

Děkuji za pozornost

matej.cajthaml@ssps.cz
ssps.cajthaml.eu



Vytvořeno pomocí aplikace Materialist